## Usage:

Java –jar target\Dependency-jar-with-dependencies.jar dataFileName.txt

Argument: “dataFileName.txt” has to be a text document. It can contain letters and numbers, which can be separated by any character in the set (“- . , ; |” or “ ” ).

Output: The result will show on the console. It is also stored in a file in the same directory as the input file. The result file name is the name of the input data file with a prefix “result_”. For example, the input data file is “SampleData.txt”. The result file will be “result_ SampleData.txt”.

## Unit tests :

3 set of data are used to test the algorithm.

- testData1 for the data under the directory(“testData/SampleData.txt”) , provided with this project.
- testData2 for the data under the directory (“testData/SampleData 2.txt”) , a loop test (“A B”, “B C”, “C D”, “D E”, “E F”, “F G”, “G A”). “A B” means A directly depends on B. “B C” directly depends on C, and so on.
- testData3 for the data under the directory (“testData/SampleData 3.txt”) , testing using different characters (“- . , ; |” or “ ” ) to split a string in the data text.

## Algorithm:

1. Read a line from the data text file, for example “A B C”, which means A directly depends on B and C. “A”, “B”, “C’ are called nodes here.
2. A hash map is used to store and search for all the nodes. Each node has a successor set, which stores all its successors and no duplicates. It also has a predecessor set, which stores all its predecessors. For example “A B C” and “B C E”, “A” is a predecessor for “B” and “C”. “B” and “C” are successors for “A”. Since “A” depends on “E” through “B”, “E” is a successor for “A” and will be put in “A”’s successor set. Vise versa, “A” is a predecessor for “E” and will be put in “E”’s predecessor set.
3. Whenever a new line is read, for example, “B C E”, the head node “B” will check if “C” and “E” are in its successor set. If not, they are the new successors and will be added to the head node’s successor set. A new set is

created to track all new-added successors for the head node. A queue is also created for exploring and storing these new-added successors. For a new successor in the queue, its successor set will also be explored to check if its successors are also in the head node's successor set. If not, its new successors will be added to the head node's successor set and the queue, too. After a member in the queue is explored, it will be removed from the queue. After all members are removed from the queue, the process of searching for new successors for the head node stops. Now we get a set which has all new-added successors for the head node. We call it, "new successor set".

4. For the head node, we explore its predecessor set. Each predecessor will add the "new successor set" to its successor set, too.
5. Each member in the "new successor set" will also add the head node and all of the head node's predecessors to its predecessor set.
6. The first 5 steps repeat until reaching the end of the input data file.

## Illustration

The input data file has the following lines. Each quotation represents a line.

"A B C", "B C E", "C G", "D A F", "E F", "F H"

The following steps show how the algorithm works after reading the first three lines.

**Initial state:** A node list is used to store all the nodes. It is empty before reading a line.

**Step 1.** After reading line "A B C", "B" and "C" are added in A's predecessor set. "A" is added into the successor set of "B" and "C".

The status of the node list: (<> **A** <B, C>) (<A> **B** <>) (<A> **C** <>)

(<> **A** <B, C>) represents node A in the node list. <> represents A's predecessor set, which is empty now. <B, C> represents A's successor set, which has "B " and "C".

(<A> **B** <>) represents node B in the node list. <A> represents B's predecessor set, which has "A". <> represents C's successor set, which is empty.

(<A> **C** <>) represents node C in the node list.

**Step 2.** After reading line “B C E”

**Step 2.1** Add “E” into the node list. “C” and “E” are added to B’s successor set. The underscore highlights the changes. “B” is added to the predecessor set of “C” and “E”.

node list: (<> **A** <B, C>) (<A> **B** <C, E>) (<A, B> **C** <>) (<B> E <>)

“new successor set” for B: <C, E>

“Queue” [C, E]

**Step 2.2** “C” is removed from “Queue”. C’s successor set is empty.

node list: (<> **A** <B, C>) (<A> **B** <C, E>) (<A, B> **C** <>) (<B> E <>)

“new successor set” for B: <C, E>

“Queue” [ E]

**Step 2.3** “E” is removed from “Queue”. E’s successor set is empty.

node list: (<> **A** <B, C>) (<A> **B** <C, E>) (<A, B> **C** <>) (<B> E <>)

“new successor set” for B: <C, E>

“Queue” []

**Step 2.4** After the “Queue” is empty, update the head node B’s the predecessor, “A”, with the “new successor set”, <C, E>. “E” is added to A’s successor set.

node list: (<> **A** <B, C, E>) (<A> **B** <C, E>) (<A, B> **C** <>) (<B> E <>)

“new successor set” for B: <C, E>

“Queue” []

**Step 2.5** Each member in the “new successor set” <C, E>, update its predecessor set with B’s predecessor set <A>. “A” is added to the predecessor set of “C” and “E”.

node list: (<> **A** <B, C, E>) (<A> **B** <C, E>) (<A, B> **C** <>) (<A, B> E <>)

“new successor set” for B: <C, E>

“Queue” []

**Step 3.** After reading line “C G”

**Setp 3.1** Add “G” into the node list. “G” is added to C’s successor set. “C” is added to the predecessor set of “G”.

node list: (<> **A** <B, C, E>) (<A> **B** <C, E>) (<A, B> **C** <G>) (<A, B> E <>)

(<C> G <>)

“new successor set” for C: <G>

“Queue” [ G]

**Step 3.2** “G” is removed from “Queue”. G’s successor set is empty.

node list: (<> **A** <B, C, E>) (<A> **B** <C, E>) (<A, B> **C** <G>) (<A, B> E <>)

(<C> G <>)

“new successor set” for C: <G>

“Queue” [ ]

**Step 3.3** update the head node C’s the predecessor, “A” and ”B”, with the “new successor set”, <G>.

node list: (<> **A** <B, C, E, G>) (<A> **B** <C, E, G>) (<A, B> **C** <G>) (<A, B> E <>)

(<C> G <>)

“new successor set” for C: <G>

“Queue” [ ]

**Step 3.4** Each member in the “new successor set” <G>, update its predecessor set with C’s predecessor set <A, B>.

node list: (<> **A** <B, C, E, G>) (<A> **B** <C, E, G>) (<A, B> **C** <G>) (<A, B> E <>)

(<A, B, C> G <>)

“new successor set” for C: <G>

“Queue” [ ]

**Step 4 :** After reading “D A F”, repeat the similar steps as the above, and so on.